STRIPS planning

- A set of *operators*, where each operator has
  - a set of *parameters*
  - a set of *preconditions*
  - a set of *effects*, consisting of *add* effects and *delete* effects
- A set of *objects* to instatiate an operator's parameters
  - a fully instantiated operator is called an *action*
- A set of propositions representing the *initial state*
- A set of propositions representing the *goals*

- **Planning problem:** Find a sequence of actions that, starting in the initial state, achieve all the goals

Simple Rocket domain

- Figure 1 from Blum and Furst 1997

Approaches to STRIPS planning

- Search through the space of *world states*
  - *forward* search, *regression* search, *bidirectional* search, *means-ends* analysis, …
- Search through the space of *plans*
  - *total order* planning or *partial order* planning
- Search through a *planning graph*

Graphplan

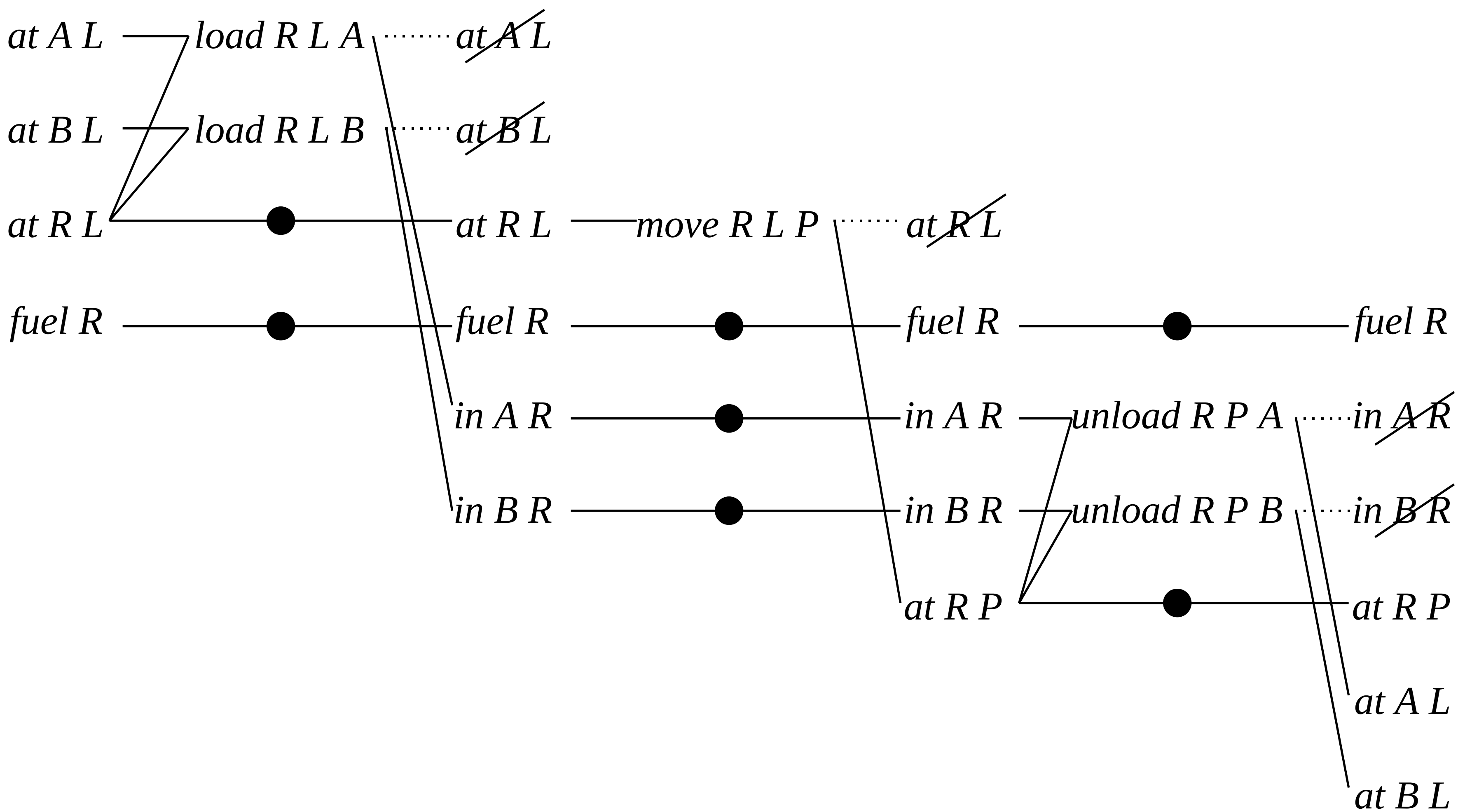
- Construct a graph that represents all valid plans up to a maximum length
- Search the graph for a valid plan

Valid plans in Graphplan

A *valid plan* is

- A set of actions
  - includes special *no-op* or *frame* actions
- Specified times for each action
  - actions at the same time don't *interfere* with each other
- Preconditions of actions at time 1 must be in the initial state
- Preconditions of actions at time $t > 1$ must be made true by the plan at time t
  - propositions true at time $t > 1$ are the effects of actions at time $t-1$
- Goals are true at the final time step

A valid plan for a Rocket problem



Planning graphs

- Like valid plans *without* the restriction that actions at the same time don't interfere with each other
- A planning graph is a leveled graph with
  - two kinds of nodes (propositions and actions)
    - alternates between proposition levels and action levels
  - three kinds of edges (preconditions, add effects, delete effects)
- An action is at action level *i* if all its preconditions are at proposition level *i*
- A proposition is at proposition level $i > 1$, if it is an add effect of some action (including *no-op* actions) at level $i–1$

Planning graph for a Rocket problem

- Figure 2 from Blum and Furst 1997

Mutual exclusions

- Two actions at the same level are mutually exclusive if no valid plan could possibly contain both of them
- Two propositions at the same level are mutually exclusive if no valid plan could possibly make themboth true

Propagating mutual exclusions

- Two actions at the same level are mutually exclusive if
  - *Interference*: if either action deletes a precondition or add effect of the other
  - *Competing needs*: if a precondition of one action and a precondition of the other action are mutually exclusive
- Two propositions at the same level are mutually exclusive if
  - all ways of creating one proposition are exclusive of all ways of creating the other proposition

Graphplan

function *Graphplan*(*Ops*, *Objs*, *InitState*, *Goals*)

    Initialize *G* with proposition level *1* using *InitState*
    for *i = 1* incrementing by *1* **do**
        if proposition level *i* contains all *Goals* **and**
            no two goals are mutually exclusive at level *i* **then**
            Search *G* for a valid plan to achieve *Goals*
            if *G* contains a valid plan **then return** the plan
        endif
        Augment *G* with action level *i* and proposition level *i+1*
        if *termination condition* is satisfied **then return** “no plan”
    endfor

end *Graphplan*

Size of the planning graph

- **Theorem:** Consider a planning problem with *n* objects, *p* propositions in the initial state, *m* operators each having a constant number of parameters. Let *l* be the length of the longest add list Then the size of a *t*-level planning graph, and the time needed to create the graph, are polynomial in *n*, *m*, *p*, *l*, and *t*

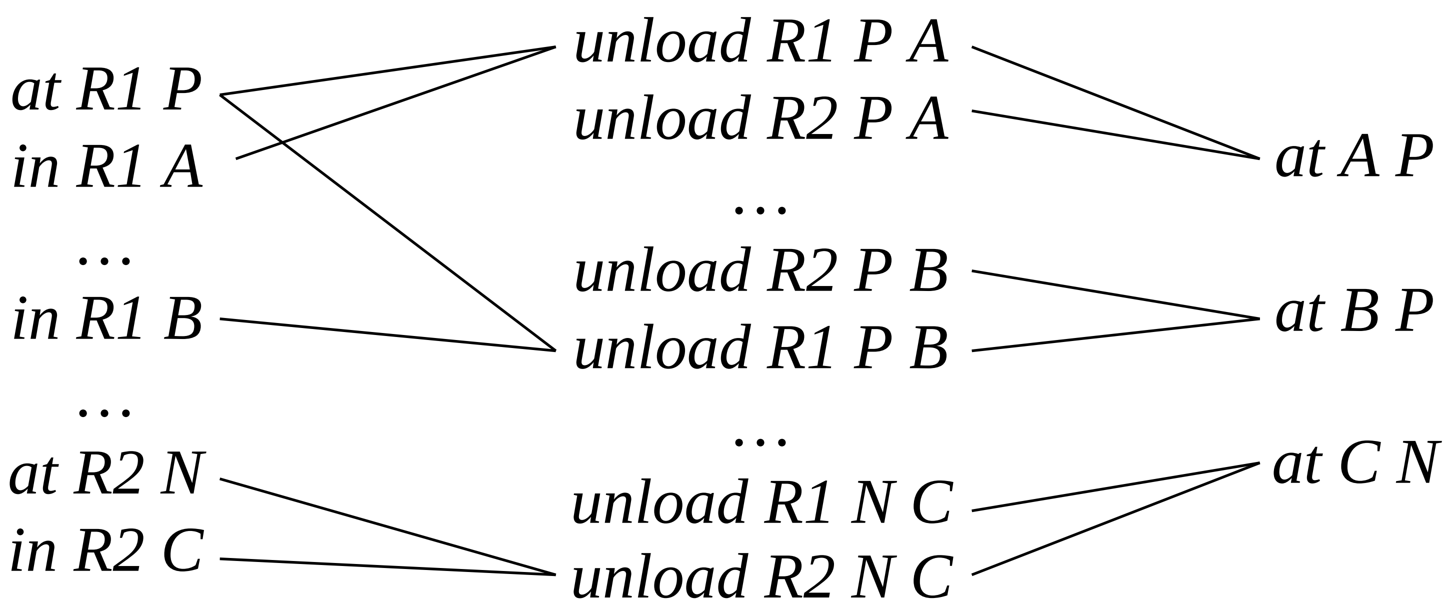
Searching the graph for a valid plan

function *Solve-goals*(*goal-set, level*)
    if *level == 1* **then**
        if *InitState*   *goal-set* **then return** ({})
    else
        if *IsMemoized*(*goal-set, level*) **then return** ()
        new-actions = Select-actions(*goal-set*, {}, *level*)
        if *new-actions* ≠ () **then**
            return *new-actions*
        else
            Memoize(*goal-set, level*)
            return ()
        endif
    endif
end *Solve-goals*

Selecting actions at a given level

```
function Select-actions(goal-set, actions, level)
    if goal-set = {} then
        return Solve-goals(preconditions(actions), level – 1)
    else
        goal = pop(goal-set)
        if goal achieved by some action in actions then
            return Select-actions(goal-set, actions, level)
        else
            for  each action  that achieves goal and
                   not mutex with any action in actions  do
                new-actions = Select-actions(goal-set, actions     {action}, level)
                if new-actions ≠ () then return new-actions
            endfor
            return ()
        endif
    endif
```

Example


at R1 P
in R1 A
…
in R1 B
…
at R2 N
in R2 C
unload R1 P A
unload R2 P A
…
unload R2 P B
unload R1 P B
…
unload R1 N C
unload R2 N C
at A P
at B P
at C N

Experimental results

Accounting for Graphplan's efficiency

- Mutual exclusions
- Consideration of parallel plans
- Memoizing
- Low-level costs

Leveling off

- **Lemma:** If no valid plan exists, then there exists a proposition level *P* such that all future proposition levels are identical to *P*
  - *i.e.*, contain the same propositions and mutual exclusions
  - graph is said to have *leveled off* after *P*
- **Corollary:** No solution exists if
  - a goal does not appear in *P* or
  - *P* has mutually exclusive goals

Termination condition

- Let S_i^t denote the set of memoized goal sets at level *i* after an unsuccessful stage *t*
- **Theorem:** If the graph has leveled off at some level *n* and a stage *t* has passed in which $| S_n^{t-1} | = | S_n^t |$, then no valid plan exists